

The Art of Software Design, a Video Game for Learning Software Design Principles

Gamification of Software Design Learning Objectives

Dave R. Stikkolorum
drstikko@liacs.nl

Michel R.V. Chaudron
chaudron@liacs.nl

Oswald de Bruin
snake.ossi@gmail.com

Institute of Advanced Computer Science, Leiden University
Niels Bohrweg 1, Leiden, the Netherlands

ABSTRACT

This paper introduces our gamification of a part of our software design curriculum. Based on typical design principles a motivating learning game is developed to train students in software design. We use Bloom's taxonomy to determine learning objectives. We keep the player engaged with direct feedback in a challenging level based game with increasing complexity. Players can evaluate their design actions with the help of the visualisation of control and data flows. The main learning objective: applying design principles, fits the game's main activity. This supports the learning by doing approach of lecturers. A user test indicates possible learning effects and a playable game.

Categories and Subject Descriptors

D.2.2 [SOFTWARE ENGINEERING]: Design Tools and Techniques—*Design Principles*; D.2.10 [SOFTWARE ENGINEERING]: Design; K.3.1 [COMPUTERS AND EDUCATION]: Computer Uses in Education—*Game Based Learning*

General Terms

Design, Didactics, Gaming

Keywords

Gamification, Video Game, Education, Software Design, Modelling

1. INTRODUCTION

Designing of software systems is one of the difficult tasks in the field of software engineering. It is no surprise that learning software design is a difficult task for students too. They have to face the challenge of abstracting structure and behaviour for possible software solutions. Most lecturers

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ModelsGamification12, October 01 - 05 2012, Innsbruck, Austria
Copyright 2012.

choose the approach of letting students practice with artificial cases and repeat this a couple of times. We can not expect that this approach always motivates students. In this paper we introduce our gamification of a part of our software engineering curriculum. We aim to motivate students to learn software design by providing them learning material in the form of an interactive computer game.

In the field of software engineering a couple of game environments were introduced with programs like BlueJ¹ and Alice² to support students in learning to program and understand the concept of object orientation. However in this type of programs only a game world is provided, you have to create the game yourself. In the field of architecture modelling Groenewegen et al. [2] introduced a game for validating architecture models. Groenewegen et al. used a board game where the actual models play a role. We also aimed to create a game where the models itself are game elements. We used a video game. As far as we know no such game in the field of software design exists.

In this paper we discuss our gamification approach, give insight in the game itself and discuss early findings based on user tests. In section 2 we describe our method, in section 3 we describe the game design. After a 'walk through' of the game in section 4 we evaluate and discuss in sections 5 and 6. Finally we conclude and propose future work in section 7.

2. METHOD

The aim of this study is to create a playable game to explore the possible support for the learning of students. It is created using an iterative approach. The authors came together periodically and discussed and improved their different ideas until there was a first version.

The learning objectives of the game are chosen using Bloom's taxonomy [4] combined with a set of general design principles [6]: coupling, cohesion, information hiding and modularity. Bloom's taxonomy is widely used by lecturers. 'Design principles' is a typical software design subject.

For further improvement we tested the game with a user test in combination with a simple version of the 'think aloud' [5] method. We simply asked the test users to tell us their thoughts while making steps in the game. We know this is no complete validation. As mentioned before there is no

¹<http://www.bluej.org>

²<http://www.alice.org>

To determine coupling within a puzzle we used a simple approach. We only used the CBO (coupling between object classes)[1] metric.

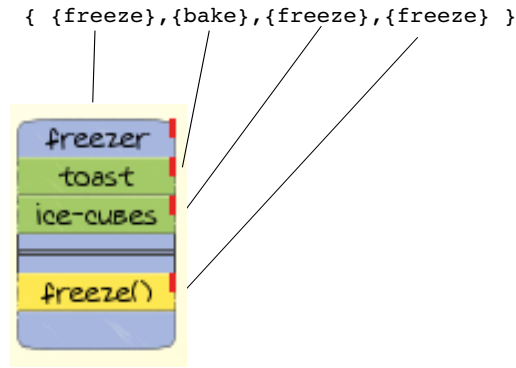


Figure 2: Connected keywords to classes, attributes and methods

CBO per class is: the number of other classes it connects to

Per design the average would be the sum of all the CBO (indexed by i) divided by the total of classes (n):

$$\text{averageCBO} = \frac{\sum_{i=1}^n \text{CBO}_i}{n}$$

3.6.2 cohesion

Every class, attribute and method has one or more keywords attached to it. We use these keywords to determine if two elements are related, if they are cohesive. This is illustrated in figure 2.

The cohesion evaluation script compares if all items of a class (including the class itself) have similar keywords. In the example provided in figure 2 only one keyword is attached per element. This can also be two or more.

Cohesion of a class (CC) is determined as follows: i. per comparison (indexed by i), divide the number of keyword matches (m) with all the other elements by the number of keywords (k) that are being considered in the comparison. This gives us the match ratio per comparison. ii. divide the sum of these ratio's by the number of comparisons (n). This gives us the total index of the similarities of a class, the cohesion:

$$CC = \frac{\sum_{i=1}^n \frac{m_i}{k_i}}{n}$$

3.6.3 information hiding and modularity

To evaluate the application of information hiding and modularity we used general design patterns. The player has only a limit of choices and is guided to a solution that uses this patterns. The evaluation script checks if the user applied the elements (classes, attributes, methods) in a way that matches the pattern.

3.7 Software Platform

The game is constructed with Gamemaker 8.1 Standard³. We made this choice so we could rapidly make fully functional prototypes. Gamemaker has a readily available engine with graphics, mouse events, scripting and other features that can be used in games.

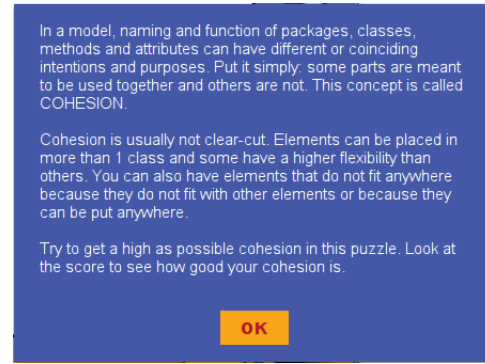


Figure 3: Pop-up with assignment for the cohesion puzzle

4. THE GAME

In this section we provide an overview of the flow of a game session. The main aim of the game is to complete every puzzle.

'The Art of Software Design'^{4,5} starts with an opening screen and gives the player the opportunity to personalise his game by entering his name. This name is used as a saved game and by clicking it, it will resume after the last finished puzzle. After entering the welcome screen a puzzle-tree appears. This tree consists of puzzles based on software design principles or combination of those principles. Players have to unlock upper level puzzles before they can do the next puzzles.

Inside a puzzle a welcome messages is showed (fig. 3). This message contains the assignment of the puzzle. Typical tasks a player has to fulfil are such as placing attributes and operations in the right classes (responsibility driven approach), connect the right classes (associate) given a typical design principle. A toolbox is present for adding these items when needed. A progress indicator shows how close the player is to the solution of the puzzle. As mentioned before the game offers the opportunity to complete a puzzle with different solutions. One can be better than the other. This is shown by the players' score as shown in figure 4.

After completing the puzzle, the player returns to the main screen where certain puzzles are unlocked. From there new challenges are possible.

The game ends when all the puzzles about basic design principles have been fulfilled. After that, in future versions of the game, series of more complex puzzles can be offered.

5. EVALUATION

In a thinking aloud setting we observed the gaming experience of a group of users and noted their comments.

We noticed that giving freedom of choice was of value. When people got stuck on a puzzle, they tried out different solutions or tried another puzzle and returned to the more difficult puzzle later on.

Players paid more attention to the instructions prior to a puzzle when there were 3 paragraphs of 4 lines of text at maximum. The text was best understood if it had an introductory paragraph, an explanatory paragraph and an

³<http://yoyogames.com/gamemaker>

⁴The Art of Software Design : <http://aosd.host22.com>

⁵Trailer : http://www.youtube.com/watch?v=xn1E2dU-_zg

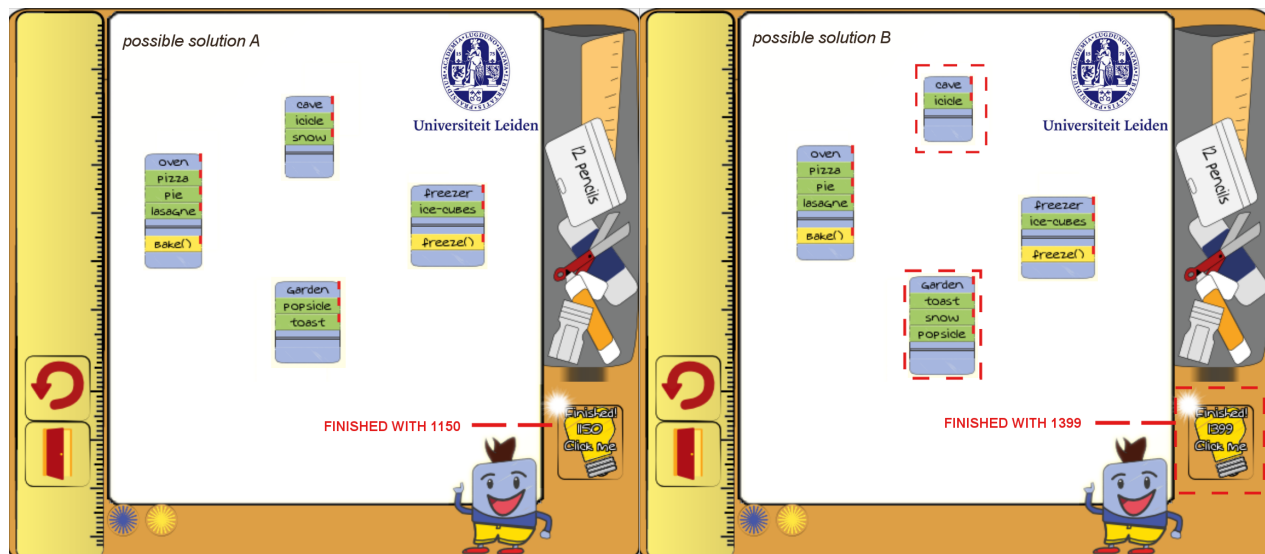


Figure 4: 2 possible solutions for the cohesion puzzle. The red striped boxes show the differences.

assigning paragraph in that order. Some puzzles seem too simple. They were solved without reading the instructions.

For us it was satisfying to see, that after a while subjects started talking about the puzzles in terms of ‘classes, methods and associations’, instead of ‘boxes, blocks and lines’, which seems to indicate some unconscious learning.

6. DISCUSSION

Determining the average coupling of the design could be a too rough measure. The average coupling can turn out to be relatively low, while a certain class in the design can have a very high coupling. It may be wise to indicate the user that the coupling of a certain class is too high.

Although we find it very plausible that we choose to simulate associative thinking with keywords in order to determine cohesion, we have no data that validates that approach. It may be wise to explore correlation between metrics such as (L)COM [3] to validate our keywords method.

We acknowledge that further study is needed to demonstrate the learning effect, but we think the unconscious learning of concepts like classes, attributes and methods at least indicates a certain learning effect.

7. CONCLUSION

The aim to create a playable learning game seems to be accomplished. Although we are not able to prove that our approach is completely valid, we have some good indicators for future research. A deeper study of the validity of the score metrics is needed. Further research is needed to demonstrate the learning effect of the game. We suggest to study both validation and learning effect in a case study that uses ‘The Art of Software Design.’

8. REFERENCES

- [1] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.
- [2] J. Groenewegen, S. Hoppenbrouwers, and E. Proper. Playing archimate models. In *Enterprise, Business-Process and Information Systems Modeling*, volume 50 of *Lecture Notes in Business Information Processing*, pages 182–194. Springer Berlin Heidelberg, 2010.
- [3] R. Harrison, S. J. Counsell, and R. V. Nithi. An investigation into the applicability and validity of object-oriented design metrics. *Empirical Software Engineering*, 3(3):255–273, 1998.
- [4] D. R. Krathwohl. A revision of bloom’s taxonomy: An overview. *Theory Into Practice*, 41(4):212–218, 2002.
- [5] C. Lewis. Using the “thinking-aloud” method in cognitive interface design. Technical report, IBM T.J. Watson Research Center, 1982.
- [6] R. C. Martin. Design principles and design patterns. *Object Mentor*, (c):1–34, 2000.
- [7] R. Moser. A fantasy adventure game as a learning environment. why learning to program is so difficult and what can be done about it. *ITiCSE 97 Proceedings of the 2nd conference on Integrating technology into computer science education*, 29(3):114–116, 1997.
- [8] A. K. Przybylski, C. S. Rigby, and R. M. Ryan. A motivational model of video game engagement. *Review of General Psychology*, 14(2):154–166, 2010.